National Snow and Ice Data Center

Supporting Cryospheric Research Since 1976

CIRES

University of Colorado **Boulder**

Completely Test-Driven

ian.truslove@nsidc.org

@iantruslove

UCAR Software Engineering Assembly, Feb 21, 2012

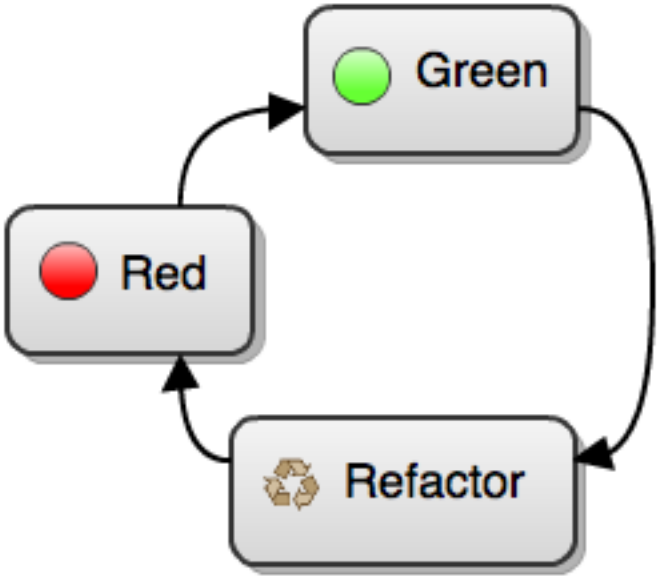
What's In It For Me?

- "So, that TDD sounds great and all, but what about <*thing*>?"
- See some techniques that really are necessary for TDD (and some others that are just cool)
- Start TDD?!

Outline

- TDD Refresher
- FIRST: Good Unit Tests
- For business's sake, AT!
- The Test Double family
- Testing the UI?
- External resources, aka "What about the DB?"
- Reality: Legacy Code
- Wrap-up

TDD Refresher



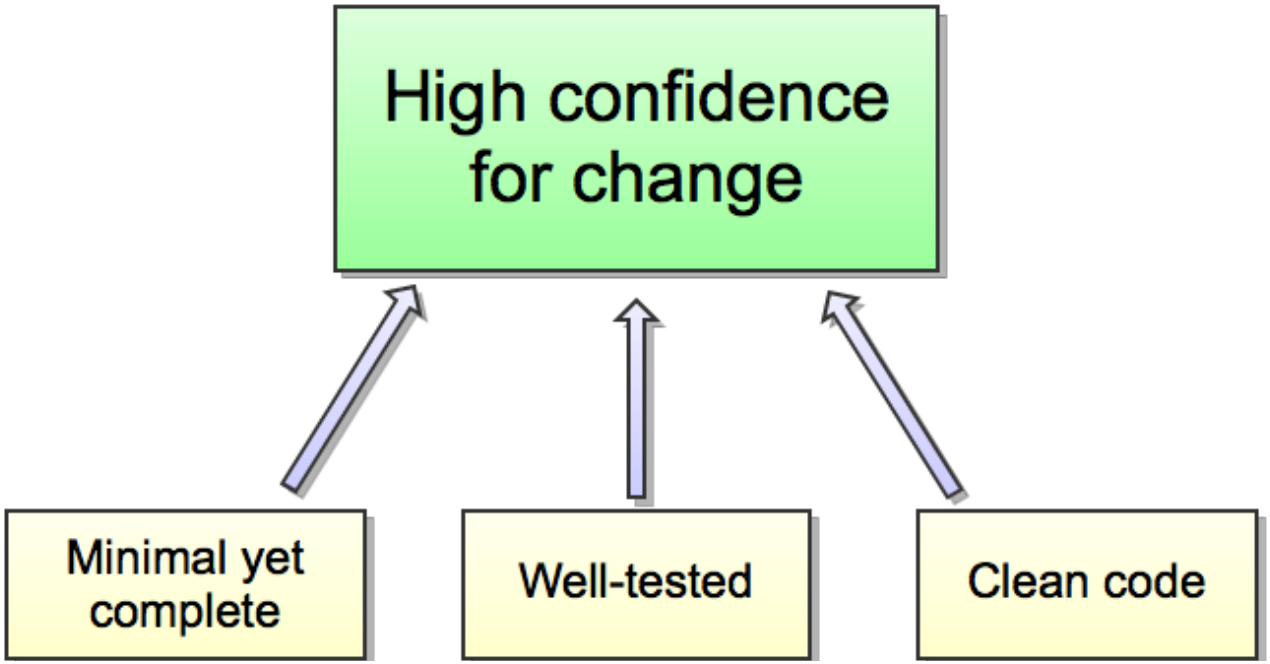
* Well, almost...

TDD Refresher

Behavior-Driven Development style

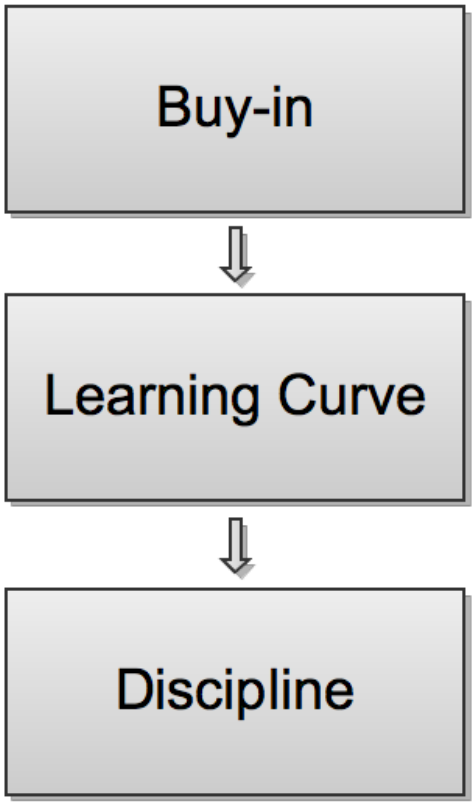
- ("should" instead of "assert")

TDD Refresher



TDD Refresher

Challenges



FIRST: Good Unit Tests

FIRST: Good Unit Tests

Fast

Independent

Repeatable

Self-verifying

Timely

FIRST: Good Unit Tests

Test behaviors, not methods

– Don’t test your privates?

FIRST: Good Unit Tests

Each test should test only one thing

- One assertion per test?

FIRST: Good Unit Tests

Only one “real” class per test

- Mock the collaborators
- But don’t mock values (e.g. java.net.URI)

FIRST: Good Unit Tests

Smell: Test code is hard to read

- *not enough refactoring*

FIRST: Good Unit Tests

Smell: Big ripples of red during changes

- *not enough refactoring, tests are too complex*

For business's sake, AT!

For business's sake, AT!

- Acceptance Criteria are specifications for what the software needs to do, written in domain language, by domain experts.
- Acceptance Tests (ATs) are executable Acceptance Criteria.
- Unambiguous, executable, repeatable specifications and documentation

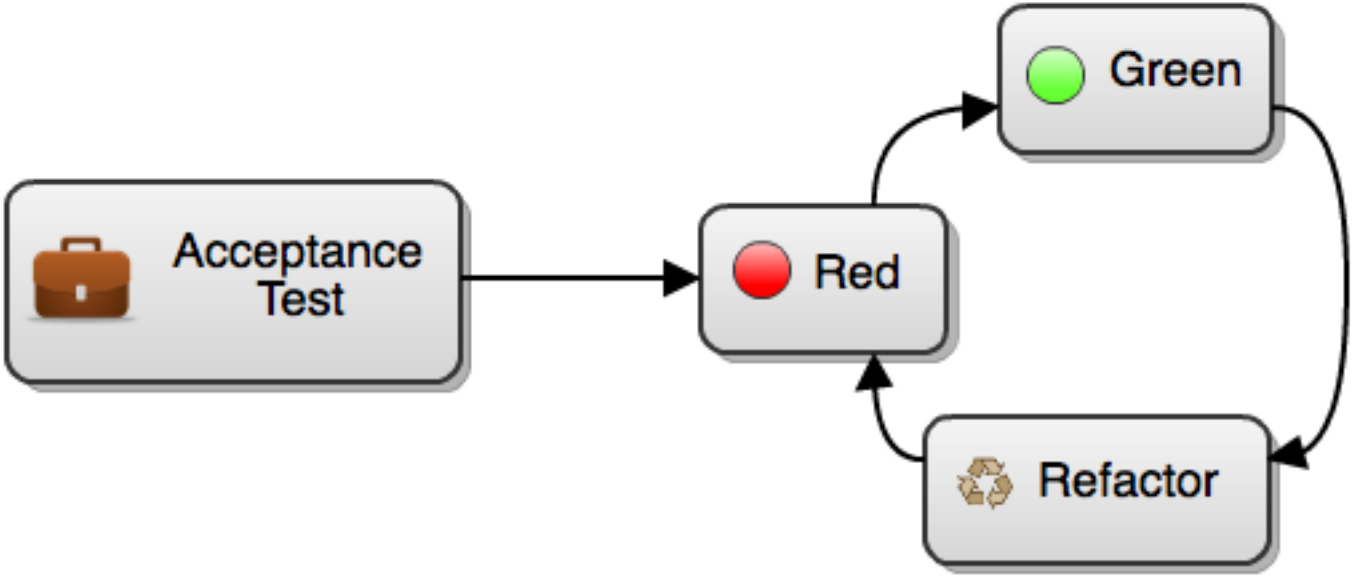
For business's sake, AT!

Given {a context}

When {an event occurs}

Then {an observable outcome}

For business's sake, AT!



Drive the TDD loop with an AT

For business's sake, AT!

Cucumber specification:

```
1 Feature: Delete a feed
2    As a Libre administrator
3    I want to delete a feed
4    So that I can remove all of its entries
5
6    Scenario: Delete feeds when the Aggregator has more than one feed
7       Given that the Aggregator has multiple feeds
8       When I delete an specific feed
9       Then the Aggregator will delete the entries from that feed
```

For business's sake, AT!

Java step implementations:

```
@Given("^that the Aggregator has multiple feeds$")
public void given_that_the_aggregator_has_aggregated_more_than_one_feed()
      throws Exception {
   steps.addInitialDataSetToSolrInstance(5);
   assertThat(
         steps.getAggregator().search().getEntries().size(),
         is(equalTo(5)));
}

@When("^I delete an specific feed$")
public void when_I_request_the_list_of_feeds() throws Exception {
   deleteSuccess = steps.getAggregator().deleteFeed(
         steps.getAggregator().listFeeds().get(0).getBaseUri().toString());
}

@Then("^the Aggregator will delete the entries from that feed$")
public void theAggregatorWillDeleteTheEntriesFromThatFeed() throws Exception {
   assertThat(deleteSuccess,is(equalTo(true)));
   assertThat(
         steps.getAggregator().search().getEntries().size(),
         is(equalTo(4)));
}
```

For business's sake, AT!

Smell: ATs with lots of technical details

- *you're using ATs instead of UTs, or not testing from the outermost point of view*

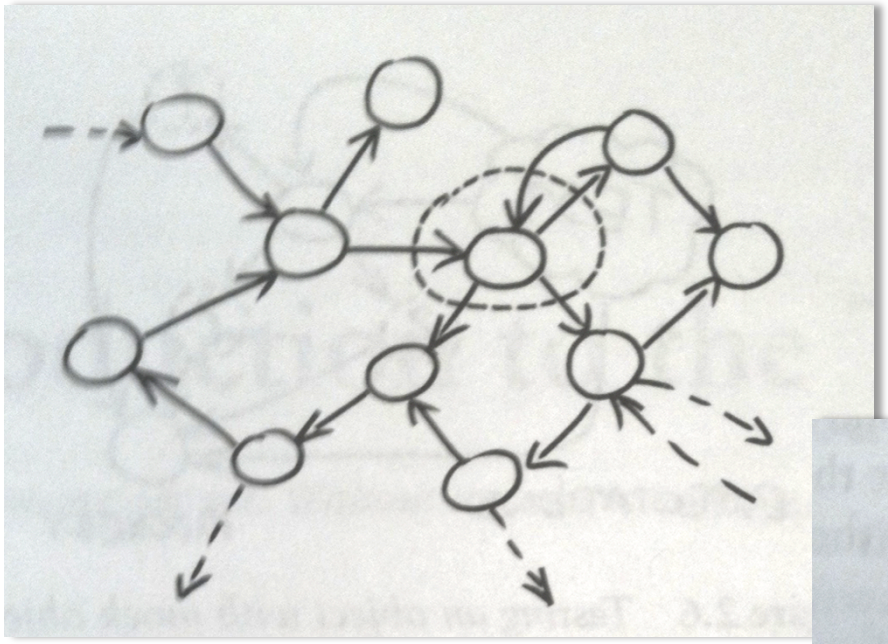
For business's sake, AT!

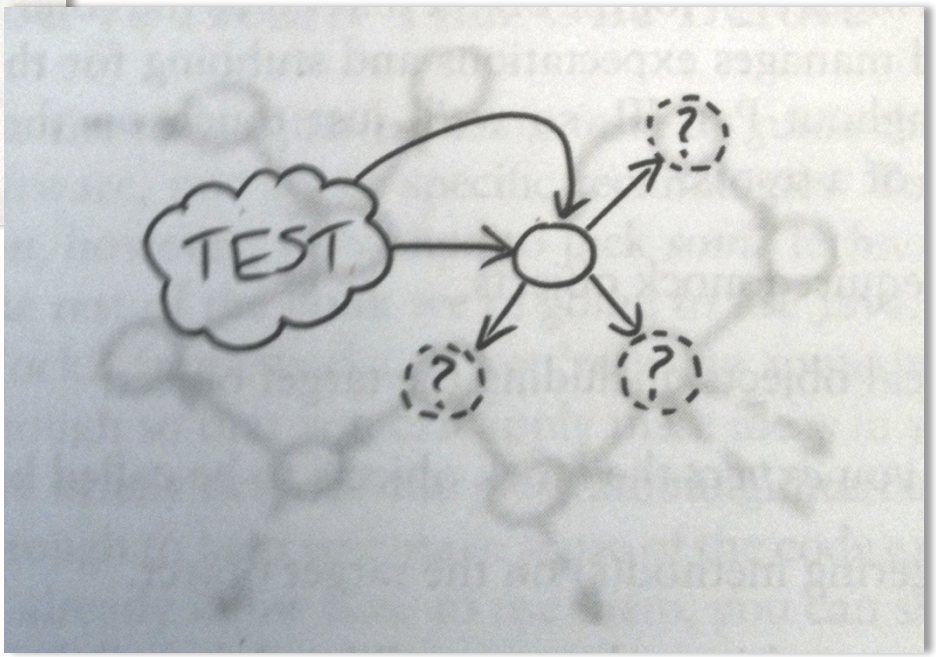
Smell: ATs keep breaking

– *not abstract enough;*
perhaps too imperative

The Test Double family

The Test Double family





The Test Double family

("Mock object" often means Test Double)

In increasing order of trickiness:

- Dummy object
- Stub
- Mock

this != Meszaros' xUnit Patterns != Sinon.JS

The Test Double family

Smell: Enormous effort setting up mock

- *pull the complexity into e.g. domain façade objects*

The Test Double family

Smell: Lots of test doubles in a test

– *poor encapsulation*

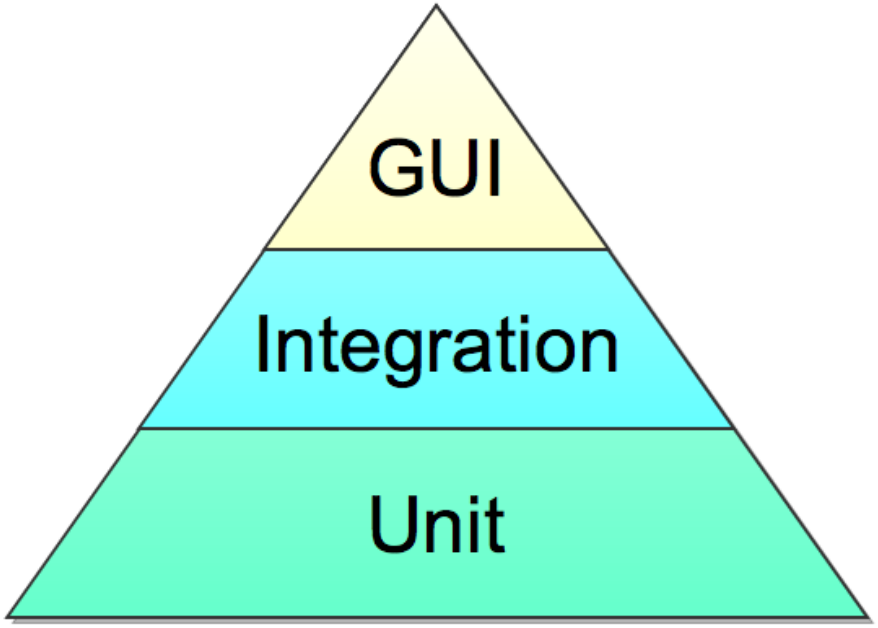
Testing the UI?

Testing the UI?

Diminishing returns: can you write an automated test to tell you whether the UI looks nice? Flows well?

Yes, but at what cost?!

Testing the UI?



Prefer unit test over integration test over AT

Testing the UI?

A strategy:

- Unit test event handlers
- Custom DSLs
- Few end-to-end ATs

Testing the UI?

Watir WebDriver – acceptance test

```
def flightsCount
  @browser.span(:text=>"IceBridge Flights") \
    .parent.parent.parent \
    .element(:css => "li.x-tree-node").count
end

def setStartDate(date)
  startDate = @browser.text_field :name => "StartDate"
  startDate.send_keys [:control, 'a'], :backspace, date, :tab
end

# snip...

it "should have at least 40 flights in the tree for 2010" do
  setStartDate '2010-01-01'
  setEndDate '2010-12-31'

  waitForResultsToAppearInFlights

  flightsCount.should be > 40
end
```

Testing the UI?

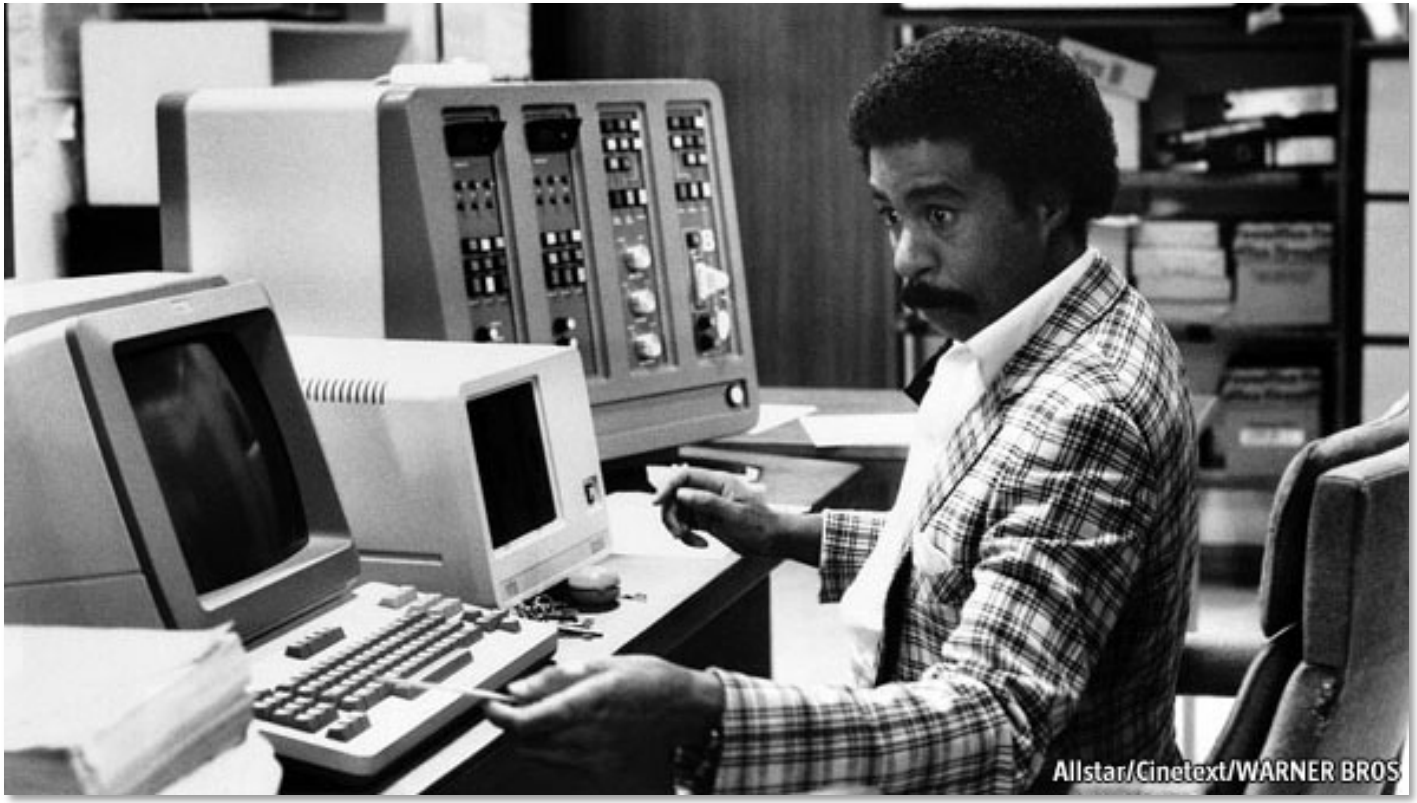
JavaScript – unit test

```
it( "contains a node indicating no matching results when search results are empty",
      function() {
   // arrange
   var osResults = createOpenSearchResultSet({total: 0, itemsPerPage: 0, startIndex:0});

   // act
   var nodeArray = this.module.buildOpenSearchTreeNodes( "providerName",
         osResults );

   // assert
   expect(nodeArray.length).toEqual(1);
   expect(nodeArray[0].text).toEqual('No matching results');
});
```

External resources, aka “What about the DB?”



External resources, aka “What about the DB?”

Mock it

Need to mock an interface,

But I bet there's no clean interface to mock

⇒ Adapt: wrap it first!

Reality: Legacy Code

Reality: Legacy Code

- It's hard. Don't start here.
- Fowler's *Refactoring* book – small, safe changes
- Wrap in tests
- Improve what you touch – "shine a light"
- 100% TDD any new code
- Feathers' *Legacy Code* book

Wrap-up

TDD Getting Started Guide™:

1. Find a Champion
2. Hit the books
3. Go slow
4. Find your frameworks
5. New project
6. Go all-out
7. Short loops
8. Refactor Mercilessly

Wrap-up

Things to read:

- *Growing Object-Oriented Software, Guided By Tests* - Freeman & Pryce
- *Test-Driven Development: By Example* – Beck
- *XUnit Test Patterns* - Meszaros
- *The RSpec Book* - Chelimsky, Astels et al
- *Clean Code: A Handbook of Agile Software Craftsmanship* - Uncle Bob
- *Test Driven: TDD and Acceptance TDD for Java Developers* – Lasse Koskela
- *Continuous Delivery* - Humble & Farley
- *Refactoring: Improving the Design of Existing Code* - Fowler
- *Working Effectively With Legacy Code* - Feathers
- Gojko Adzik's myriad web papers on automated testing

Questions

Online at http://bit.ly/yravMa or
http://dl.dropbox.com/u/4292130/UCAR_SEA_20120221/CompletelyTestDriven.pdf